A Description of R Functions Used in Analysis of Internal Migration and Post-Great Recession Recovery in Minnesota Counties and Neighborhoods

Scott Chase
Faculty Mentor Professor Zack W. Almquist

University of Minnesota - Twin Cities

Fall 2014

1 Introduction

The national economic downturn of 2008 has imparted profound effects on nearly all aspects of the national economy. An area that received considerable impact from the recession is household wealth, hypothesized to have drawn much of its change from patterns of internal migration. While it is generally accepted in the fields of demography and sociology that a relationship between migration and concentration of wealth exists, the extent to which household wealth is impacted remains unknown. In this report I will include and describe the R code I have contributed to an ongoing project titled "Internal Migration and Post-Great Recession Recovery in Minnesota

Counties and Neighborhoods." This project, which seeks to examine the relationship between household wealth and internal migration during the recent recession, is being conducted under the supervision of University of Minnesota Department of Sociology and School of Statistics Professor Zack W. Almquist.

Economist Valeria Groppo of the German Institute for Economic Research argues that "global internal mobility is increasing." Although disagreement over the rate at which internal migration is increasing often occurs, scholars in the areas of demography and sociology typically agree with Groppo when she asserts that "overall internal migration rates are increasing."[1] Focusing on the recession starting in 2008, which spans the time period subject to Groppo's analysis, the Federal Reserve Bank of St. Louis estimates that a staggering 45% net loss in household wealth occurred in the United States.[2] Coupled with evidence provided by Groppo, this information merits examination of the existence of a relationship between household wealth and migration. Demographers and sociologists have examined the impact of household wealth and inequality on migration, [3] but an analysis of the reverse can provide additional, valuable information. This project seeks to examine

[1]Valeria Groppo, "Internal Migration in Developing Countries." DIW Berlin, July 29, 2014, Accessed December 19, 2014, `http://www.diw.de/de/diw_01.c.471438.de/internal_migration_in_developing_countries.html.`

[2]"How much Household Wealth Has Been Recovered?" *Federal Reserve Bank of St. Louis.* May 2013. Accessed 23 Nov 2014 `www.stlouisfed.org/publications/ar/2012/pages/ar12_2j.cfm`

[3]David McKenzie and Hillel Rapoport. "Network effects and the dynamics of migration and inequality: theory and evidence from Mexico." *Journal of development Economics* 84, no. 1 (2007): 1-24.

the relationship between internal migration in the United States and household wealth in recent years to provide crucial information to policymakers as to where economic recovery initiatives would prove most effective.

The demographic and spatial data currently available holds the key to unlocking this relationship and the extent to which it reaches. Annually, the Internal Revenue Service (IRS) releases a wealth of county-to-county migration data. The information currently available spans the years 1990-2011, recording the number of people moving from any given county in the US to another. In 1990 and 2000 the US Census Bureau released a similar record of migration counts, but only at the county level. In 2010 the US Census Bureau provided the ACS, which can be aggregated in five-year increments to build similar county level estimates. However, scholars have long struggled with the issue of working with administrative data and connecting it to meaningful existential categories. [4] [5]

In order to prepare for analysis, we have concentrated our efforts to build neighborhood level estimates of migration flows for the entire US. Seeking to bridge the gap in the detail of information, this project seeks to integrate both IRS and US Census migration data at the finely grained 2000 Census level to approximate a more detailed migration record for the entire 20 year interval from 1990 to 2010.

[4] Pitsillidis et al. "Botnet Judo: Fighting Spam with Itself." In *NDSS*. 2010.

[5] Weber, Ingmar, and Emilio Zagheni. "Studying inter-national mobility through IP geolocation." In *Proceedings of the sixth ACM international conference on Web search and data mining,* pp. 265-274. ACM, 2013.

Although my work on this project remains far from complete and analysis of the results is impending, I will give a description of the functions in the statistical software program R that I have authored thus far, including functions to collect demographic data from the 2000 and 2010 US Census servers, a similar function to collect ACS data, and code used to create a database of migration counts based on IRS data. Within the upcoming months, these functions will be published in the R packages "UScensus2000" and "UScensus2010," maintained by Dr. Almquist. In addition to a final analysis, the R code produced provides a convenient method for future demographic researchers to utilize US Census and ACS demographic and economic data.

As part of the effort to estimate neighborhood level migration flows, I have written a family of related functions to collect and organize user specified data from the 2000 and 2010 US Census databases and the ACS. All of these functions require the user to input several similar arguments. First, the functions require a Federal Information Processing Series (FIPS) code for the particular state the user wishes to collect data for. The FIPS code, defined by the National Institute of Standard and Technology, serves as an identifier of a geographic area for ease of use with information systems.[6] As a result, FIPS codes are used in accessing the Census Bureau Application Programming Interface (API) and to organize information within the data sets that these functions extract. The Census API is described as a method "to

[6] "Fips General Information." *National Institute of Standards and Technology.* Accessed December 1, 2014. `http://www.nist.gov/itl/fipsinfo.cfm`

allow software applications to communicate with each other accurately and securely over the Internet in a simpler way." [7] This systematic method of accessing the API requires the user to select relevant demographic data requires the use of a character vector containing Census Bureau variable identifiers. Each of these refers to information such as population of an area, number of African-American males, or median age. A useful feature of these functions is in the large number of demographic variables that can be extracted from the US Census Bureau's database. The user can specify any number of desired characteristics, and the functions will return the appropriate variables. In addition to the flexibility of the number of possible variables, a convenient option exists for the user to select a level of geographic specification such as a county or tract - for which the functions will extract data from the US Census servers. The federal government partitions the entire United States into levels in order from largest to smallest of state, county, tract, block group, and block - with a block being roughly the size of a neighborhood in an urban setting. These functions create the potential for analysis to range from a broader county level to the finely grained neighborhood level. Such a range of versatility will prove itself very useful to future demographic research. As this is a significant undertaking, the Census Bureau keeps records of which users are accessing their data through API keys, a unique string

[7] Abdulrahman Ruqayya et al. "Data Extraction from Online Social Networks Using Application Programming Interface in a Multi Agent System Approach." In *Transactions on Computational Collective Intelligence XI,* pp. 88-118. Springer Berlin Heidelberg, 2013.

of characters issued to one individual. Required for use of the Census Data and ACS functions, these keys can be obtained for free upon agreement to the US Census Bureau's terms of service. See the end of this document for instructions on obtaining one.

The central procedure of the 2000 and 2010 Census data functions focuses on filling in a template with demographic information. Based on the arguments the user inputs, it loads the necessary R packages from the Comprehensive R Archive Network (CRAN.) CRAN is defined by Kurt Hornik, the author of a significant amount of R language documentation as "a collection of sites which carry identical material, consisting of the R distributions, contributed extensions, and documentation for R." [8] Also loaded are spatial data sets currently maintained by Dr. Almquist to create a "template" of FIPS codes and empty cells corresponding to the amount of data requested. The function then repeatedly calls the US Census API to fill in this template with the selected demographic data for each FIPS code at the specified level, resulting in a convenient, searchable list of all data requested by the user. These functions return the newly created data sets within the R session, in addition to automatically saving them to the user's current local directory.

As the 2000 US Census function is mainly just an analog to the 2010 case with some slight modifications to account for differences between the 2010 Census API and the 2000 Census API, only the 2010 Census Function will

[8]Hornik, Kurt. "The R FAQ." Last modified October 26, 2014. Accessed December 18, 2014. `http://CRAN.R-project.org/doc/FAQ/R-FAQ.html`

be outlined here.

2 2010 Census Data

```
CensusData2010<-function(varlist,fipscode,
level=c("county","tract","block group","block","cdp"),key)
```

I define the function header in R as "CensusData2010," with the first of the arguments being "varlist," a character vector of the identifiers for the demographic variables the function will return. Each element in this vector refers to a US Census demographic variable. For instance, "P0010001" is total population of the geographic area. It is worthwhile to note that the 2000 US Census typically defines its variable identifiers with one fewer character, for instance the designator for total population in the 2000 Census is "P001001." Using the 2000 Census designators with the 2010 Census function will result in errors, and vice versa. The variable identifiers must be formatted as a character vector, as follows.

```
varlist<-c("P0010001","P0030002","P0030003")
```

After the function header is defined, it requires R package "UScensus2010" loaded in the current session, as the spatial data sets it contains are used as a template of FIPS codes and empty cells for which to place the appropriate data. The selected level, which is not case sensitive, must be one of the given choices of federal-government-defined geographic areas, county, tract, block

group, block, or cdp. In the event that it is not one of these choices, an error will be given. The "fipsdatabase" object created provides use for two purposes. First, it determines whether the state FIPS code input is valid, and if it is not, results in an error. Its second use lies in relating the name of the state to its FIPS code.

```
suppressMessages(require("UScensus2010"))
level<-tolower(level)
level<-match.arg(level,several.ok=FALSE)
data(countyfips)
fipsdatabase<-as.data.frame(unique(cbind(countyfips$statename,
substr(countyfips$fips,1,2))),stringsAsFactors=F)

   if(nchar(fipscode)!=2||!any(grepl(fipscode,fipsdatabase[,2])==T))
      stop("Must be a valid State FIPS code")
```

In order to systematically and uniformly extract data from the Census Bureau's servers, the method of calling the API requires linking the FIPS code template and the information the user input. To accomplish this, an inner function, defined within the "CensusData2010" function, named "API-call" is defined. As before, it ensures the validity of the FIPS code and level the user inputs. Likely for privacy reasons, the US Census Bureau API will not allow users to obtain data in a single API call for all of the smaller geographic levels, such as block and block group, in an entire state. The API will only allow users to select data for every block group in a specific county, and will only allow the user to pull out data at the block level if a tract is specified. In order to circumvent the Census Bureau's restrictions, it is

necessary to undertake multiple API calls to gather all of the demographic data for each state. The function will halt and give the user errors if the FIPS code given does not indicate a small enough geographic area.

```
APIcall<-function(vars,fips,lev = c("county","tract","block
    group","block","cdp"),key)
{
   {
   lev<-tolower(lev)
   lev<-match.arg(lev,several.ok=FALSE)
   suppressMessages(require(rjson))
   if(level=="block group"&&nchar(fips)<5)
      stop("FIPS must be 5 digits for Block Group, in the order of
          state, county.")
   if(level=="block"&&nchar(fips)<11)
      stop("FIPS must be 11 digits for Block, in the order of
          state, county, tract.")
```

For each combination of demographic variable identifiers, level, and state there exists a unique URL, which can be obtained by systematically parsing the relevant information together. Upon accessing an appropriate URL, the Census Bureau will provide the demographic data for the area that the URL corresponds to. In order to input the arguments from the "CensusData2010" function into the inner function "APIcall," the object "vars" will take the same value as "varlist" the character vector containing demographic variable identifiers. For the same purpose, "fips" takes the same value as "fipscode." The API key originally input to the "CensusData2010" function is automatically input into this inner function. Note that this step will not work without Internet access.

```
if(lev=="cdp")
      url<-paste("http://api.census.gov/data/2010/sf1?get=",gsub(",
          ",",",toString(vars)),"&for=place:",substr(fips,3,7),
"&in=state:",substr(fips,1,2),"&key=",key,sep="")

   else if(lev=="county")
      url<-paste("http://api.census.gov/data/2010/sf1?get=",gsub(",
          ",",",toString(vars)),"&for=county:",substr(fips,3,5),
"&in=state:",substr(fips,1,2),"&key=",key,sep="")

   else if(lev=="tract")
      url<-paste("http://api.census.gov/data/2010/sf1?get=",gsub(",
          ",",",toString(vars)),"&for=tract:",substr(fips,6,11),
"&in=state:",substr(fips,1,2),"+county:",substr(fips,3,5),"&key=",
key,sep="")

   else if(lev=="block group")
   url<-paste("http://api.census.gov/data/2010/sf1?get=",gsub(",
       ",",",toString(vars)),"&for=block+group:",substr(fips,12,12),
"&in=state:",substr(fips,1,2),"+county:",substr(fips,3,5),"+tract:",
substr(fips,6,11),"&key=",key,sep="")

   else if(lev=="block")
   url<-paste("http://api.census.gov/data/2010/sf1?get=",gsub(",
       ",",",toString(vars)),"&for=block:",substr(fips,12,15),
"&in=state:",substr(fips,1,2),"+county:",substr(fips,3,5),"+tract:",
substr(fips,6,11),"&key=",key,sep="")
```

Upon creation of a valid URL, the "CensusData2010" function accesses, downloads, and places appropriately the demographic data corresponding to this URL. With the use of the "rjson" package in R, and some fairly simple formatting, this function creates and returns a data frame containing the information requested by the user. The column names of the data frame correspond to the selection of demographic variables and the FIPS codes

of the specified geographic area. In order to make searching for a specific FIPS code easier in analysis, every row name is a condensed FIPS code, with no spaces or dashes. The "CensusData2010" function also ensures the formatting of the demographic variables as numeric, as opposed to a string or factor.

```
   document <- fromJSON(file=url)
   m<-matrix(unlist(document),nc=length(document[[1]]),byrow=TRUE)
   #Formatting of the Data, making sure the demographic variables
      are reported numerically, not as factors
   colnames(m)<-m[1,]
   m<-rbind(m[2:NROW(m),])
   m<-as.data.frame(m,stringsAsFactors=FALSE)
   for(i in 1:length(vars))
      m[,i]<-as.numeric(m[,i])
   #Naming the rows a condensed version of the FIPS
   start<-length(vars)+1
   stp<-ncol(m)
   d<-NULL
   for(k in 1:dim(m)[1])
   {
      for(j in start:stp)
      {
         d<-paste(d,m[k,j],sep="")
      }
      rownames(m)[k]<-d
      d<-NULL
   }
   return(m)
}}
```

After the inner function "APIcall" accesses and downloads the requested data, the necessity lies in formatting and returning the data in a convenient fashion. To accomplish this, the "APIcall" function calls the Census Bu-

reau API repeatedly and systematically as it iterates through the FIPS code template, and returns the requested data for the specified level. The function then places the returned demographic data appropriately in the FIPS code template. The next steps will require a specific R package based on which geographic level is selected. The "CensusData2010" proceeds under the assumption that the functions to load the spatial data templates from the "UScensus2010" family of R packages have been run. For an explanation and commentary, see the end of this document. The data in the collection of "UScensus2010" R packages are named systematically so loading the FIPS code templates automatically is possible with the use of the "fipsdatabase" object created earlier for relating the name of the state to its FIPS code. For instance, the data for Maine at the county level is named "maine.county10," and likewise the data for Minnesota at the tract level is "minnesota.tract10." In order to inform the user of the pace at which the function is running, a "z" counter is defined and prints the number of completed API calls in varying intervals. Because there are different numbers of each geographic area in each state, the "z" counter takes different values to accommodate this.

```
data(states.names)
if(level=="tract")
  {
    lst<-paste(states.names,".tract10",sep="")
    suppressMessages(require("UScensus2010tract"))
    z=100
  }
if(level=="county")
  {
```

```
      lst<-paste(states.names,".county10",sep="")
      suppressMessages(require("UScensus2010county"))
      z=25
   }
if(level=="block group")
   {
      lst<-paste(states.names,".blkgrp10",sep="")
      suppressMessages(require("UScensus2010blkgrp"))
      z=200
   }
if(level=="block")
   {
      lst<-paste(states.names,".blk10",sep="")
      suppressMessages(require("UScensus2010blk"))
      z=1000
   }
if(level=="cdp")
   {
      lst<-paste(states.names,".cdp10",sep="")
      suppressMessages(require("UScensus2010cdp"))
      z=50
   }
```

The FIPS code template loaded belongs to a much larger set of spatial data, and must be broken down and appropriately formatted in order to assign the data accessed from the API to the proper location in the data frame. This spatial data comes preloaded with a specific set of census spatial and demographic data, which may or may not align with what the user requests, so this pre-loaded data is erased and replaced with empty cells to be filled in during the iteration of the "APIcall" function. Based on the number of demographic variables the user requests, the appropriate number of columns in the template to fit the selected data will be kept and the rest

erased. The column names will be assigned as the appropriate geographic level for identification, with each column containing a complete list of FIPS codes for the selected level and state. The other columns are named as the US Census-defined variable identifiers, with each element in these columns referring to the value of the demographic variable for the area specified by the FIPS code in the same row. The value of "n" marks the end of the FIPS codes for each row, and the start of the demographic data provided. Due to the different format of the spatial data at the CDP level, additional formatting is necessary.

```
statename<-fipsdatabase[which(fipsdatabase[,2]==fipscode),1]
d<-lst[which(grepl(statename,lst)==T)]
d<-as.data.frame(get(data(list=d)))
d<-as.data.frame(d,stringsAsFactors=FALSE)
if(level=="tract")
   {
      d=d[,1:(4+length(varlist))]
      n=5
      colnames(d)<-c("State","County","Tract","FIPS",c(varlist))
   }
if(level=="county")
   {
      d=d[,3:(5+length(varlist))]
      n=4
      colnames(d)<-c("State","County","FIPS",c(varlist))
   }
if(level=="block group")
   {
      d=d[,1:(5+length(varlist))]
      n=6
      colnames(d)<-c("State","County","Tract","Block
          Group","FIPS",c(varlist))
   }
```

```
if(level=="block")
   {
      d=d[,1:(6+length(varlist))]
      n=7
      colnames(d)<-c("State","County","Tract","Block
          Group","Block","FIPS",c(varlist))
   }
if(level=="cdp")
   {
      d<-d[,1:(4+length(varlist))]
      n=5
      d[,1]<-d[,2]
      d[,2]<-d[,3]
      d[,3]<-d[,4]
      d[,4]<-paste(d[,2],d[,3],sep="")
      colnames(d)<-c("Name&Designation","State","Place","FIPS",c(varlist))
   }
```

The "CensusData2010" function iterates through the empty template of FIPS codes created, using the condensed FIPS codes in the empty template and the level, variable identifiers, and Census API key originally input to the "CensusData2010" function as arguments for each iteration of the "APIcall" function. The appropriate demographic data is returned and assigned to the corresponding empty cells in the empty FIPS code template.

An issue that merited attention in the early testing phase of the "CensusData2010" function involved the function running very quickly, resulting in the function rapidly hitting the Census API with data requests. At some unpredictable point, the Census Bureau's API restrictions would not allow the data to be extracted. To circumvent this problem, in the event that the API does not return data, the "APIcall" function waits 15 seconds and

	State	County	Tract	FIPS	P0010001	P0030002	P0030003	P0030004	P0030005
1	23	003	952900	23003952900	NA	NA	NA	NA	NA
2	23	003	950900	23003950900	NA	NA	NA	NA	NA
3	23	003	952100	23003952100	NA	NA	NA	NA	NA
4	23	003	952300	23003952300	NA	NA	NA	NA	NA
5	23	003	951200	23003951200	NA	NA	NA	NA	NA
6	23	003	951300	23003951300	NA	NA	NA	NA	NA

Figure 1: Empty Template of Maine FIPS Codes at the Tract Level

	State	County	Tract	FIPS	P0010001	P0030002	P0030003	P0030004	P0030005
1	23	003	952900	23003952900	2958	2847	3	29	22
2	23	003	950900	23003950900	2069	2029	10	10	4
3	23	003	952100	23003952100	2768	2698	5	26	9
4	23	003	952300	23003952300	2652	2485	5	113	3
5	23	003	951200	23003951200	2549	2451	15	20	21
6	23	003	951300	23003951300	3386	3270	12	42	18

Figure 2: Filled in Template of Maine FIPS Codes at the Tract Level

repeats the same API call. If the API returns the data, the FIPS template is filled in appropriately. If the API call fails to extract data a second time, that particular row is filled with NAs and the function prints a warning to notify the user of where the error is located.

```
d[,n:(n+length(varlist)-1)]<-NA
rownames(d)<-NULL
data<-NULL
fips1<-fipscode
for(j in 1:dim(d)[1])
{  data<-try(APIcall(varlist,$FIPS[j],lev=level,key),silent=TRUE)
   if(class(data)=="try-error"
   {
      Sys.sleep(15)
      data<-try(APIcall(varlist,d$FIPS[j],lev=level,key),silent=TRUE)
      if(class(data)=="try-error")
         {
            data<-NULL
            warning("NAs generated at row ",print(j),". Sorry!")
            d[j,n:(n+length(varlist)-1)]=NA
         }
      else
         d[j,n:(n+length(varlist)-1)]=data[,1:length(varlist)]
```

```
    }
    else
       d[j,n:(n+length(varlist)-1)]=data[,1:length(varlist)]
    if(j%%z==0)
       print(j)
  }
  if(level!="block group")
       label<-level
  if(level=="block group")
    label<- blockgroup "
```

Next, the "CensusData2010" function automatically saves the data frame as an .rda file, a file containing a single object in the R console, allowing for the object to be used in a different R session.

```
for(z in n:dim(d)[2])
{
   d[,z]<-as.numeric(d[,z])
}
savename<-paste(statename,"_",label,"_2010",sep="")
assign(savename,d)
save(list=savename,file=paste(statename,"_",label,"_2010",".rda",sep=""))
return(d)
}
```

To ensure that the "CensusData2010" function does not print the entirety of the data frame to the R console, it is worthwhile to instruct the function to assign the returned data frame to some variable in the current R session. Examples of the function calls and the first few rows of output are given below.

```
mainecdp<-CensusData2010(varlist<-c("P0010001","P0030002",
"P0030003","P0030004","P0030005"),
fipscode="23",level="cdp",key)

mainecounty<-CensusData2010(varlist<-c("P0010001","P0030002",
"P0030003","P0030004","P0030005"),
fipscode="23",level="county",key)
```

	State	County	FIPS	P0010001	P0030002	P0030003	P0030004	P0030005
1	23	019	23019	153923	146802	1159	1809	1422
2	23	029	23029	32856	30257	140	1603	149
3	23	003	23003	71870	68759	455	1225	312
4	23	009	23009	54418	52741	221	220	446
5	23	007	23007	30768	29952	71	116	132
6	23	025	23025	52228	50733	192	241	295

Figure 3: Maine County Data

	Name&Designation	State	Place	FIPS	P0010001	P0030002	P0030003	P0030004	P0030005
1	Presque Isle city	23	60825	2360825	9692	9155	61	232	85
2	Caribou city	23	10565	2310565	8189	7865	35	113	56
3	Eastport city	23	21730	2321730	1331	1224	10	48	6
4	Calais city	23	09585	2309585	3123	2981	17	40	19
5	Old Town city	23	55225	2355225	7840	7302	70	129	139
6	Ellsworth city	23	23200	2323200	7741	7482	51	34	82

Figure 4: Maine CDP Data

3 ACS Data

In contrast to the "CensusData2010" and "CensusData2000" functions, I have written a separate function titled "getACSdata" to extract data from the American Community Survey administered by the Census Bureau. The ongoing ACS strives to provide communities with relevant information about age, sex, education, income, health insurance, and other information to assist

in planning investments and services. [9] In contrast to the decennial Census, the Census Bureau conducts the ACS each year and provides estimates and standard errors for relevant information. Although the basic procedure of the "getACSData" function is similar to the procedure of the "CensusData2010" function, some slight differences appear, and thus I will outline it here. Initially, it requires the "UScensus2010" R package from CRAN, and creating "fipsdatabase" to ensure the validity of the FIPS code. For convenience, as the variables for the 2010 ACS 5-year-estimates are associated with a standard error for each estimate, the user need only enter the actual variable identifiers. The function determines the standard error identifiers for each estimate and reports these along with the estimates themselves. Because the ACS only provides its estimates at the county level, the "getACSdata" function loads and formats the same spatial data templates at the county level, and places the returned demographic data in appropriate location following the same procedure as the "CensusData2010" function.

```
getACSdata<-function(vars,fips,key)
{  suppressMessages(require("UScensus2010"))
   data(countyfips)
   fipsdatabase<-as.data.frame(unique(cbind(countyfips$statename,
substr(countyfips$fips,1,2))),stringsAsFactors=F)
   if(nchar(fips)!=2||!any(grepl(fips,fipsdatabase[,2])==T))
      stop("Must be a valid State FIPS code")
   evars<-rep(0,length(vars))
   for(s in 1:length(vars))
      evars[s]<-paste(substr(vars[s],1,nchar(vars[2])-1),"M",sep="")
```

[9] "About the American Community Survey." United States Census Bureau. Accessed 13 Dec. 2014 http://www.census.gov/acs/www/about_the_survey/american_community_survey/

```
   varlist<-as.vector(rbind(vars,evars))
   data(states.names)
   lst<-paste(states.names,".county10",sep="")
   suppressMessages(require("UScensus2010county"))
   statename<-fipsdatabase[which(fipsdatabase[,2]==fips),1]
   d<-lst[which(grepl(statename,lst)==T)]
   d<-as.data.frame(get(data(list=d)))
   d<-as.data.frame(d,stringsAsFactors=FALSE)
   d=d[,3:(5+length(varlist))]
   d[,4:(4+length(varlist)-1)]<-NA
   rownames(d)<-NULL
   colnames(d)<-c("State","County","FIPS",c(varlist))
```

The inner function "ACS_APIcall" is designed to deal with the simpler ACS county level estimates, and therefore remains much shorter. Only the "UScensus2010" and "UScensus2010county" packages are required for accessing the county level spatial data templates.

```
ACS_APIcall<-function(vars,fips,key)
   {
      suppressMessages(require(rjson))
url<-paste("http://api.census.gov/data/2010/acs5?get=",gsub(",
    ",",",toString(vars)),"&for=county:",
substr(fips,3,5),"&in=state:",substr(fips,1,2),"&key=",key,sep="")
   document <- fromJSON(file=url)
   m<-matrix(unlist(document),nc=length(document[[1]]),byrow=TRUE)
   colnames(m)<-m[1,]
   m<-rbind(m[2:NROW(m),])
   m<-as.data.frame(m,stringsAsFactors=FALSE)
   for(i in 1:length(vars))
      m[,i]<-as.numeric(m[,i])
      start<-length(vars)+1
   stp<-ncol(m)
   d<-NULL
   for(k in 1:dim(m)[1])
   {
      for(j in start:stp)
```

```
    {
       d<-paste(d,m[k,j],sep="")
    }
    rownames(m)[k]<-d
    d<-NULL
  }
  return(m)
  }
```

Analogously to the "CensusData2010" function, the "getACSdata" function iterates through the empty FIPS code template, using both the information provided by the user and the information contained in each row of the template to call the API. If in the event that the function runs too fast and the API denies the request for data, it waits 15 seconds and tries again. If the API denies the request on the second attempt, the "getACSdata" function generates NAs at that particular row and prints a warning notifying the user of where the errors occurred. The data frame containing the demographic information is organized and saved as an .rda file for further use. Below are examples of a function call and the first few rows of data extracted.

```
maineACS<-getACSdata(vars=c("B01001_002E","B01001_026E"),
fips="23",key)
```

	State	County	FIPS	B01001_002E	B01001_002M	B01001_026E	B01001_026M
1	23	019	23019	75294	140	77640	140
2	23	029	23029	16303	47	16851	47
3	23	003	23003	35607	59	36805	59
4	23	009	23009	26557	108	27752	108
5	23	007	23007	15069	91	15588	91
6	23	025	23025	25902	189	26359	189

Figure 5: Maine ACS Data

4 IRS Migration Data

As part of the effort to estimate neighborhood level migration flows, I have accessed and conveniently formatted a complete record of (IRS) migration flows. The focus of the IRS migration data centers on building a complete, searchable set of data, as opposed to creating functions to continually access data: the objective of the functions to access the US Census Data and the ACS. Annually, the IRS releases a wealth of information in county-to-county migration flows each year based on tax returns, although not in a consistent or necessarily convenient format for each year. When used in conjunction with the family of functions to access US Census data and used to create a detailed estimate of US migration at the neighborhood level over the past 20 years, analysis can begin.

The IRS releases its migration data in various formats, such as Microsoft Excel and CSV files, all of which required the writing of a unique method to organize in a consistent format. As a result of these efforts there now exists a complete record of county-to-county migration flows, within a single, 9-million element matrix for each year from 1993 to 2011. Every county in the United States has one corresponding row and column. Any given element in the matrix indicates the number of people moving from the county the row corresponds to and into the county the column corresponds to. Each matrix also reports marginal entries, such as those moving to and from a foreign country, or between regions in the United States. Although unique parsers

had to be written for each year, the general procedure was the same. All of the parsers iterated through each list of migration counts and organized them in a format compatible with the code used to organize the data into matrices. I will give an overview of the code used to create the matrices, as well as examples of select parts of the matrices. It is unnecessary to run the code outlined here; the matrices are already built.

A single argument to the IRS migration function is required: a "datafile." In addition to indicating the name of the file containing the migration data, this also specifies for which tax season the matrix is to be created. As the geographic boundaries of counties in the United States sometimes undergo changes between each decennial census, a different FIPS code database was created for each decade to more accurately reflect migration counts. This function also creates marginal entries to record the migration counts between US counties and foreign countries, between regions within the US, such as west, southwest, midwest, or northeast, between different states in the same region, or between counties in the same state.

```
national_matrix<-function(datafile)
{
    regionaldatabase<-readRDS("~/USmigrationData/Data/regionaldatabase.rds")
regionfips=as.data.frame(cbind(c("Northeast","Midwest","South","West"),
c("59001","59003","59005","59007")),stringsAsFactors=FALSE)
   names(regionfips)=c("Region","FIPS")
      if(substr(datafile,6,6)=="1")
         {
         fipsdatabase<-readRDS("~/USmigrationData/Data/FIPS
             Database/countyfips2010.rds")
         }
```

```
    else if(substr(datafile,6,6)=="0")
      {
      fipsdatabase<-readRDS("~/USmigrationData/Data/FIPS
          Database/countyfips2000.rds")
      }
    else if(substr(datafile,6,6)=="9")
      {
      fipsdatabase<-readRDS("~/USmigrationData/Data/FIPS
          Database/countyfips1990.rds")
      }
  fips<-sort(unique(fipsdatabase[,1]))
  states<-(unique(fipsdatabase[,4]))
  mat<-matrix(nrow=(length(fips)+4),ncol=(length(fips)+4))
```

The migration data as read from the parsers is organized in a list of states. Each state has two elements, an inflow component containing counts of people moving into each county in the state, and an outflow component containing counts of people moving out of each county in the state. The "national_matrix" function iterates through each state's inflow and outflow components and places the migration count in the corresponding element of the matrix. This function will also determine and report the counts for the regional, foreign, and same state counts using the "regionaldatabase" and "regionfips" objects, created to link state-to-state migration with regional migration.

```
  colnames(mat)<-c(fips,"SS","SR","DR","F")
  rownames(mat)<-c(fips,"SS","SR","DR","F")
  mat[is.na(mat)]<-0
  datafile<-gsub("\\.rda$","",datafile)
  loaded_data <- load(paste("~/USmigrationData/Data/IRS Migration
      Data/", datafile, ".rda", sep=""))
  d<-get(loaded_data[1])
```

```
   midwestfips="59003"
   northeastfips="59001"
   southfips="59005"
   westfips="59007"
   for(k in 1:length(states))
   {  state<-states[k]
      inflow<-as.data.frame(d[[which(names(d)==tolower(state))]][1])
      outflow<-as.data.frame(d[[which(names(d)==tolower(state))]][2])
      inflow<-na.omit(inflow)
      outflow<-na.omit(outflow)
      thisregion<-regionaldatabase[which(regionaldatabase[,2]==state),4]
      thisregion<-regionfips[which(regionfips$Region==thisregion),2]
      other.regions=regionfips[c(which(regionfips[,2]!=thisregion)),2]
      for(i in 1:dim(inflow)[1])
      {#Regional
      if(paste(inflow[i,3],inflow[i,4],sep="")==
"59001"||paste(inflow[i,3],inflow[i,4],sep="")==
"59003"||paste(inflow[i,3],inflow[i,4],sep="")==
"59005"||paste(inflow[i,3],inflow[i,4],sep="")=="59007")

      {#Other Region
      if(any(other.regions==paste(inflow[i,3],inflow[i,4],sep="")))
         mat[which(rownames(mat)=="DR"),which(colnames(mat)==
paste(inflow[i,1],inflow[i,2],sep=""))]<-as.numeric(inflow[i,7])+
mat[which(rownames(mat)=="DR"),
which(colnames(mat)==paste(inflow[i,1],inflow[i,2],sep=""))]

      #Same Region
      if(!any(other.regions==paste(inflow[i,3],inflow[i,4],sep="")))
         mat[which(rownames(mat)=="SR"),which(colnames(mat)==
paste(inflow[i,1],inflow[i,2],sep=""))]<-as.numeric(inflow[i,7])+
mat[which(rownames(mat)=="SR"),
which(colnames(mat)==paste(inflow[i,1],inflow[i,2],sep=""))]
        }

      #Regular Migration
         mat[which(rownames(mat)==paste(inflow[i,3],inflow[i,4],sep="")),
which(colnames(mat)==paste(inflow[i,1],inflow[i,2],sep=""))]
<-as.numeric(inflow[i,7])
```

```
      #Foreign
      if(inflow[i,3]=="98"&&inflow[i,2]!="000")
         mat[which(rownames(mat)=="F"),
which(colnames(mat)==paste(inflow[i,1],inflow[i,2],sep=""))]
<-as.numeric(inflow[i,7])

      #Same State
      if(inflow[i,3]=="97"&&inflow[i,4]=="001")
         mat[which(rownames(mat)=="SS"),
which(colnames(mat)==paste(inflow[i,1],inflow[i,2],sep=""))]
<-as.numeric(inflow[i,7])+mat[which(rownames(mat)=="SS"),
which(colnames(mat)==paste(inflow[i,1],inflow[i,2],sep=""))]

         }
```

This portion of the "national_matrix" function only handles the inflow migration data; an analogous version follows this in the actual code to handle the outflow migration data. The function saves the newly created matrix as an .rda file and returns it in the current R session. Below is an example of the first 10 rows and columns of the matrix containing the migration counts for 2007-2008.

	01001	01003	01005	01007	01009	01011	01013	01015	01017	01019
01001	16791	NA	NA	NA	NA	NA	NA	NA	NA	NA
01003	22	56982	NA	NA	NA	NA	NA	25	NA	NA
01005	NA	NA	8651	NA	NA	14	NA	NA	NA	NA
01007	NA	NA	NA	6533	NA	NA	NA	NA	NA	NA
01009	NA	NA	NA	NA	17089	NA	NA	NA	NA	NA
01011	NA	NA	17	NA	NA	3103	NA	NA	NA	NA
01013	NA	NA	NA	NA	NA	NA	7189	NA	NA	NA
01015	NA	20	NA	NA	11	NA	NA	40201	NA	37
01017	NA	NA	NA	NA	NA	NA	NA	NA	12358	NA
01019	NA	NA	NA	NA	NA	NA	NA	43	NA	7899

Figure 6: Preview of National Matrix

The cell in the first row and first column indicates that 16,791 people living in county "01001" - designating Autauga County, Alabama - moved,

but stayed within the county. It displays that 43 people moved from county "01019" to county "01015," and 37 people moved from county "01015" to county "01019." Each element with an NA means that there were no recorded migrations between those two counties, or that there were too few to be reported by the IRS for privacy reasons. Note that the matrix is not symmetric, as it reports inflow and outflow migration counts for each county in the United States, and these are not necessarily the same for each county. The diagonal of the each matrix designates the number of people that moved to a different location within the same county.

	01001	01003	01005	01007	01009	01011	01013	01015	01017	01019
56033	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
56035	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
56037	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
56039	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
56041	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
56043	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
56045	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
SS	1309	1808	251	436	1119	123	236	1275	402	212
SR	397	930	NA	NA	NA	NA	NA	512	NA	NA
DR	222	720	NA	NA	NA	NA	NA	312	NA	NA
F	53	51	NA	NA	NA	NA	NA	41	NA	NA

Figure 7: Preview of National Matrix Residuals

The matrix displays the marginal entries for each row and column along the bottom and right edges of the matrix. As the bottom few rows indicate, 1309 people moved from elsewhere in Alabama into county "01001," Autauga County. 397 people moved from a different location in the same region, but not the same state, into Autauga County. 222 people moved from a different region into Autauga County, and 53 people move from a foreign country into Autauga County. The analogous elements are stored on the other end of the matrix, which would indicate how many people moved from Autauga

county from a foreign country, from a different region, from somewhere else in the same state, or from somewhere else in the same region, but not the same state. A helpful mnemonic for remembering the interpretation of these matrices is "from row, to column," meaning that for any given element of the matrix, the count reported is the amount of people coming from the county that the row corresponds to, and going to the county that the column corresponds to.

As analysis is forthcoming, this is intended to be a thorough description of the functions that were created for analysis of this project. The functions will be published in the R packages maintained by Dr. Almquist in upcoming months, and will be available for free, public use.

5 Notes

5.1 Installer Functions

In order to access the spatial data templates for the "getACSdata" function, "CensusData2000" function, and the "CensusData2010" function, it is necessary to load the spatial data. A large portion of the data is too big to store on CRAN, so it is stored in a remote repository. It can be installed for all of the levels provided, using the appropriate commands. Note that the user will have to specify their operating system, "osx" for Macintosh OSX, "windows" for Windows, and "linux" for Linux. At the block and block group levels in particular, the required data amounts to several gigabytes. I

recommend installing this data in a location with access to ample time and secure internet access.

```
install.county("osx")
install.tract("osx")
install.cdp("osx")
install.blk("osx")
install.blkgrp("osx")
```

5.2 Remote Servers

When extracting data at the block group and block levels using the "CensusData2000" and "CensusData2010" functions, a very large number of API calls are typically required, as there can be more than 200,000 blocks in a single state. In instances such as these, it is in the user's best interests to run the function on a remote server using a file sharing service, such as GitHub. Not only can this save the user significant amounts of time, but it prevents the user from having to run these functions for several hours on a personal computer.

5.3 Census Bureau API Keys

To obtain your own, individualized US Census Bureau API key, copy and paste the following URL into your web browser:

```
http://api.census.gov/data/key_signup.html
```

On the web page, enter your organization, email address, then read and agree to the Census Bureau's terms of service. The Census Bureau should send you an email containing your personal key. Note that you must click the link contained in the email to activate your key. It is convenient to save the key as an object in the current R session named "key," and then save to an .rda file. This allows the user to quickly load the .rda file and just pass in "key" as an argument to the functions that require an API key, as opposed to copying and pasting the API key each time the function is called.

References

[1] Abdulrahman Ruqayya, Daniel Neagu, D. R. W. Holton, Mick Ridley, and Yang Lan. "Data Extraction from Online Social Networks Using Application Programming Interface in a Multi Agent System Approach." In *Transactions on Computational Collective Intelligence XI,* pp. 88-118. Springer Berlin Heidelberg, 2013.

[2] "How much Household Wealth Has Been Recovered?" *Federal Reserve Bank of St. Louis.* May 2013. Accessed 23 Nov 2014. `www.stlouisfed.org/publications/ar/2012/pages/ar12_2j.cfm`

[3] Groppo, Valeria. "Internal Migration in Developing Countries." DIW Berlin. July 29, 2014. Accessed December 19, 2014. `http://www.diw.de/de/diw_01.c.471438.de/internal_migration_in_developing_countries.html`.

[4] Hornik, Kurt. "The R FAQ." Last modified October 26, 2014. Accessed December 18, 2014. `http://CRAN.R-project.org/doc/FAQ/R-FAQ.html`

[5] McKenzie, David, and Hillel Rapoport. "Network effects and the dynamics of migration and inequality: theory and evidence from Mexico." *Journal of development Economics* 84, no. 1 (2007): 1-24.

[6] "Fips General Information." *National Institute of Standards and Technology.* Accessed December 1, 2014. `http://www.nist.gov/itl/fipsinfo.cfm`

[7] Pitsillidis, Andreas, Kirill Levchenko, Christian Kreibich, Chris Kanich, Geoffrey M. Voelker, Vern Paxson, Nicholas Weaver, and Stefan Savage. "Botnet Judo: Fighting Spam with Itself." In *NDSS.* 2010.

[8] "About the American Community Survey." United States Census Bureau. Accessed 13 Dec. 2014. `http://www.census.gov/acs/www/about_the_survey/american_community_survey/`

[9] Weber, Ingmar, and Emilio Zagheni. "Studying inter-national mobility through IP geolocation." In *Proceedings of the sixth ACM international conference on Web search and data mining,* pp. 265-274. ACM, 2013.